User forms and control in Excel VBA

Objectives: At the end of this lesson you shall be able to
- **define forms and controls in VBA**
- **describe the types of excel forms**
- **describe the properties, methods and events of forms.**

Introduction to Forms and Controls

A form is a document designed with a standard structure and format that makes it easier to enter, organize, and edit information. Forms contain labels, textboxes, drop down boxes and command buttons too.

By using forms and the many controls and objects that you can add to them, you can significantly enhance data entry on your worksheets and improve the way your worksheets are displayed.

Types of Excel forms

There are several types of forms that you can create in Excel: data forms, worksheets that contain Form and ActiveX controls, and VBA UserForms.

Data form

A data form provides a convenient way to enter or display one complete row of information in a range or table without scrolling horizontally. You may find that using a data form can make data entry easier than moving from column to column when you have more columns of data than can be viewed on the screen. Excel can automatically generate a built-in data form for your range or table.

Worksheet with Form and ActiveX controls

A worksheet can be considered to be a form that enables you to enter and view data on the grid.

For added flexibility, you can add controls and other drawing objects to the worksheet, and combine and coordinate them with worksheet cells. For example, you can use a list box control to make it easier for a user to select from a list of items. Or, you can use a spin button control to make it easier for a user to enter a number.

You can display or view controls and objects alongside associated text that is independent of row and column boundaries without changing the layout of a grid or table of data on your worksheet. Many of these controls can also be linked to cells on the worksheet and do not require VBA code to make them work. For example, you might have a check box that you want to move together with its underlying cell when the range is sorted. However, if you have a list box that you want to keep in a specific location at all times, you probably do not want it to move together with its underlying cell.

Creating VBA Forms

A VBA form can be created from the code window. To create a Form in VBA,click on Insert menu in the code window and then click 'UserForm'. A UserForm1 appears in the project window.

When you create or add a form, a module is also automatically created for it. To access the module associated with a form, you can right-click the form and click View Code.Double Clicking on the Form or pressing F7 will also open the Code window. Using Shift F7 will again switch back to the Design Window.

The design time properties of the Form can be set by right clicking on the form and selecting 'Properties'. The same can be achieved by Clicking "F4" or the properties button on the Form.Controls can be placed on the form from the ToolBox as per requirement.

In addition, Controls can be added on the Form programmatically / at run time using the "Add" method. Similarly the controls can be removed from the form at run time / programmatically using the "Remove" method. As an example to add a checkbox control, we can write

Set cb1 = Controls.Add("Forms.CheckBox.1")

Some of the events and methods connected with the form object are:

Events, Activate, Deactivate, Add Control, Remove Control, Click, DblClick, Initialize, KeyPress, Resize, Scroll, Terminate, Zoom etc.

Methods Copy, Paste, Hide, Move, Print Form, Repaint, Scroll, Show etc .

The code needed to perform various operations on Forms is given in Table 1.

A sample Form for data entry of students' details, marks and results is shown in Fig. 1.

Necessary code can be attached to the Command Buttons and other controls shown. After the user enters the data, the total is calculated and the result is displayed. The records can then be stored appropriately.

Copyright Free under CC BY Licence

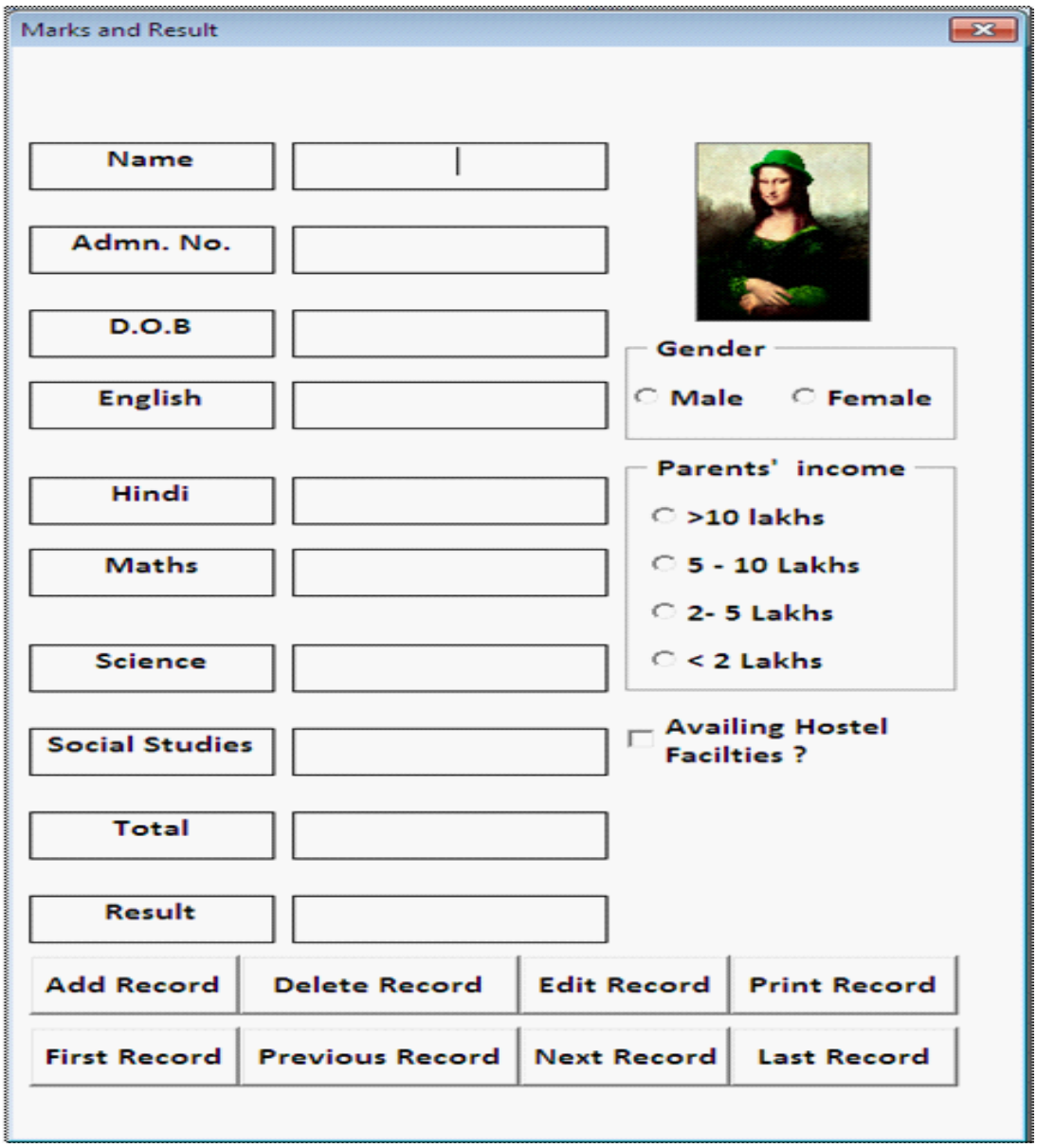
Table 1

Userform Application	VBA Code	Action
To Display a UserForm	UserForm1.Show	Displays the UserForm with name UserForm1. This code should be inserted in a Standard VBA Module and not in the Code Module of the UserForm. You can create a button in a worksheet, then right click to assign macro to this button, and select the macro which shows the UserForm.
Load a UserForm into memory but do not display	Load UserForm1	Load statement is useful in case of a complex UserForm that you want to load into memory so that it displays quickly on using the Show method, which otherwise might take a longer time to appear.
Remove a User Form from memory / Close UserForm	Unload UserForm1	Note: The Hide method (UserForm1.Hide) does not unload the UserForm from memory. To unload the UserForm from memory, the Unload method should be used.
	Unload Me	Use the Me keyword in a procedure in the Code Module of the UserForm.
Hide a UserForm	UserForm1.Hide	Using the Hide method will temporarily hide the UserForm, but will not close it and it will remain loaded in memory.
Print a UserForm	UserForm1.PrintForm	The PrintForm method sends the UserForm directly for printing.
Display UserForm as Modeless	UserForm1.Show False	If the UserForm is displayed as Modeless, user can continue working in Excel while the UserForm continues to be shown. Omitting the Boolean argument (False or 0) will display the UserForm as Modal, in which case user cannot simultaneously work in Excel. By default UserForm is displayed as Modal.
Close a UserForm	Unload UserForm1	The Unload method closes the specified UserForm.
	Unload Me	The Unload method closes the UserForm within whose Code Module it resides.
	End	Use the End statement in the "Close" CommandButton to close the form. The "End" statement unloads all forms.
Specify UserForm Caption	UserForm1.Caption = "Bio Data"	Caption is the text which describes and identifies a UserForm and will display in the header of the Userform.
Set UserForm size	UserForm1.Height = 250	Set Height of the UserForm, in points.
	UserForm1.Width = 350	Set Width of the UserForm, in points.
Set UserForm Position:		
Left & Top properties	UserForm1.Left = 30 UserForm1.Top = 50	Distance set is between the form and the Left or Top edge of the window that contains it, in pixels.
Move method	UserForm1.Move 200, 50	Move method includes two arguments which are required - the Left distance and the Top distance, in that order.

Copyright Free under CC BY Licence

Necessary code can be attached to the Command Buttons and other controls shown. After the user enters the data, the total is calculated and the result is displayed. The records can then be stored appropriately.

Fig 1



Copyright Free under CC BY Licence

Methods and Events in VBA

Objectives: At the end of this lesson you shall be able to
- **explain VBA methods and events.**

Methods and Events

Methods

A method is an action you perform with an object. A method can change an object's properties or make the object do something.

For example painting is a Method, building a new room is a method in building a new house.

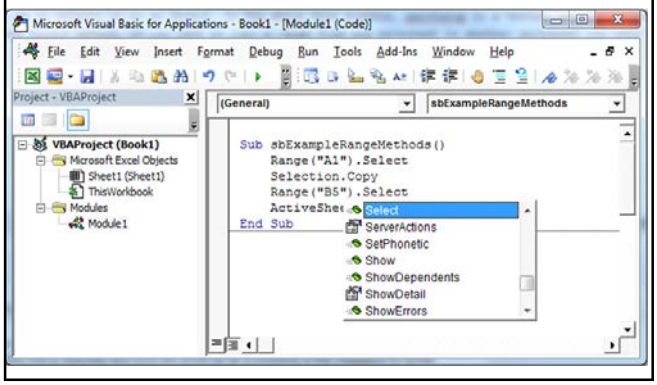
Similarly, if you want to select a range, you need Select method. If you want to copy a range from one worksheet to another worksheet you need Copy method to do it.

The following example Copies the data from Range A1 to B5.

Enter the following code in the Module1 as shown in Fig 1

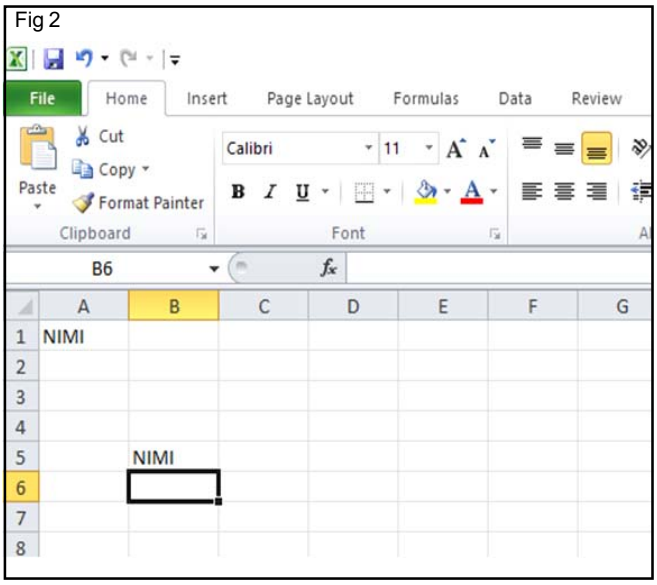
```
Sub sbExampleRangeMethods()
Range("A1").Select
Selection.Copy
Range("B5").Select
ActiveSheet.Paste
End Sub
```

Fig 1



If the above code is executed the content of cell A1 is copied to Cell B5 as shown in the Fig 2.

Fig 2



Events

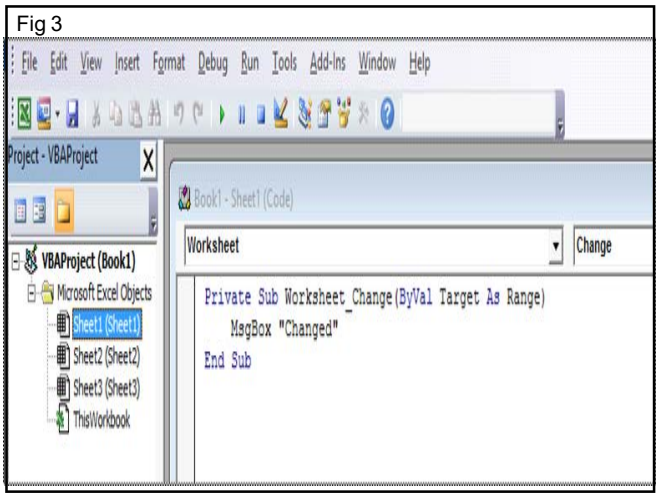
An Event is an action initiated either by user action or by other VBA code. An Event Procedure is a Sub procedure that you write, according to the specification of the event, that is called automatically by Excel when an event occurs. For example, a Worksheet object has an event named Change. If you have properly programmed the event procedure for the Change event, Excel will automatically call that procedure, always named Worksheet_Change and always in the code module of the worksheet, whenever the value of any cell on the worksheet is changed by user input or by other VBA code (but not if the change in value is a result of a formula calculation). You can write code in the Worksheet_Change event procedure to take some action depending on which cell was changed or based upon the newly changed value.

Enter the following code in the Worksheet_Change event as shown in Fig 3.

```
Private Sub Worksheet_Change(ByVal Target
As Range)
MsgBox "Changed"
End Sub
```

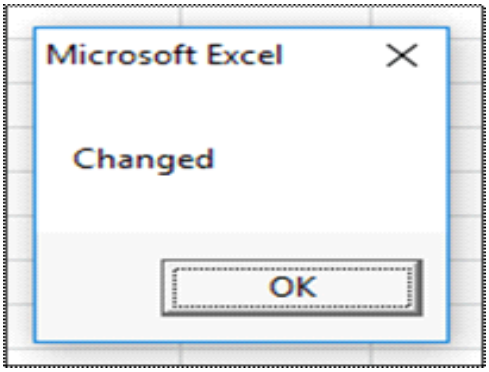

Copyright Free under CC BY Licence

Fig 3



When we change content of any Cell the following message will be displayed as shown in Fig 4.

Fig 4



Copyright Free under CC BY Licence

Debugging Techniques in VBA

Objectives: At the end of this lesson you shall be able to
- **explain about VBA debugging**
- **explain how to set and clear the Breakpoints**
- **describe use of immediate window**
- **explain about watch window.**

VBA Debugging

In Excel 2010, VBA's debugging environment allows the programmer to momentarily suspend the execution of VBA code so that the following debug tasks can be done:

1 Check the value of a variable in its current state.

2 Enter VBA code in the Immediate window to view the results.

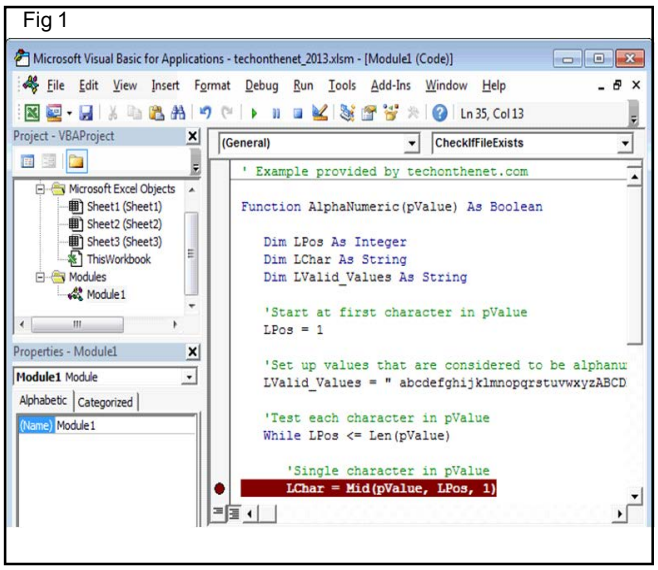
3 Execute each line of code one at a time.

4 Continue execution of the code.

5 Halt execution of the code.

These are just some of the tasks that you might perform in VBA's debugging environment. (Fig 1)

Fig 1



Breakpoint in VBA

In Excel 2010, a breakpoint is a selected line of code that once reached, the program will momentarily become suspended. Once suspended, and to use VBA's debugging environment to view the status of program, step through each successive line of code, continue execution of the code, or halt execution of the code.

And create as many breakpoints in the code as you want. Breakpoints are particularly useful when suspend the program where you suspect a problem/bug exists.

Setting a Breakpoint

First, you need to open the VBA environment. The quickest way to do this is by pressing Alt+F11 while the Excel database file is open.

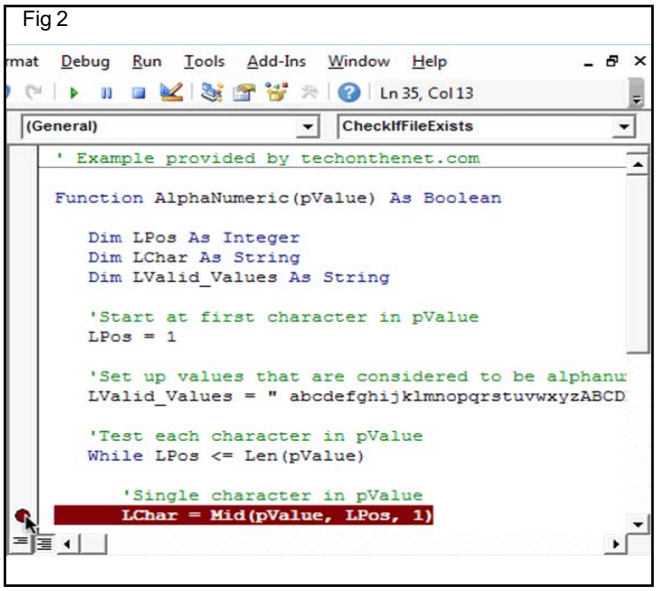
To set a breakpoint, find the line of code where to suspend your program. Left-click in the grey bar to the left of the code. A red dot should appear and the line of code should be highlighted in red.

Clear Breakpoint in VBA

A breakpoint in VBA is indicated by a red dot with a line of code highlighted in red.

To clear a breakpoint in Excel 2010, left-click on the red dot next to the line of code that has the breakpoint. (Fig 2)

Fig 2



In this example, we want to clear the breakpoint at the following line of code:

LChar = Mid(pValue, LPos, 1) (Fig 3)

Now, the breakpoint is cleared and the line of code should look normal again. (Fig 4)

Copyright Free under CC BY Licence

Fig 3

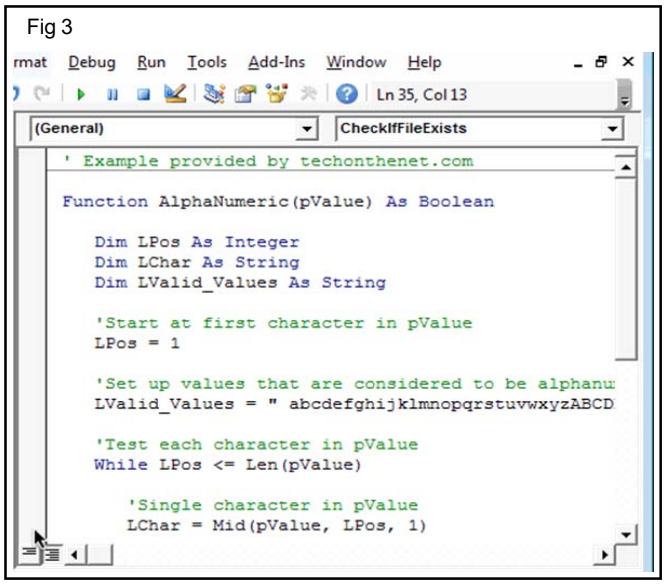
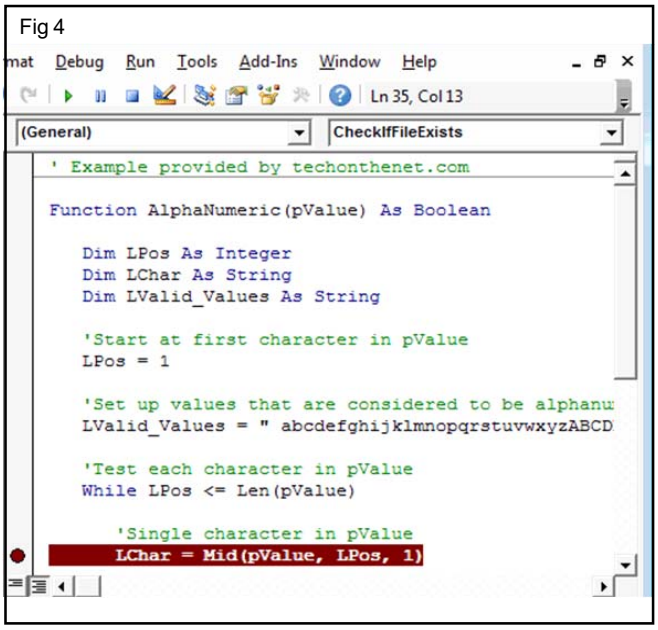


Fig 4



In this example, we've created a breakpoint at the following line of code:

LChar = Mid(pValue, LPos, 1)

Now, the breakpoint is cleared and the line of code should look normal again

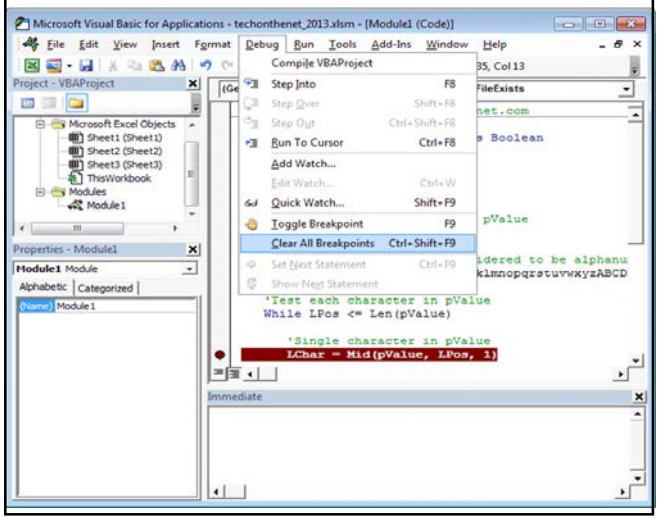
Clearing all Breakpoints

If user use as many breakpoints as you want in Excel 2010, and can save time by clearing all breakpoints in the VBA code at once.

To clear all breakpoints in the program, select "Clear All Breakpoints" under the Debug menu. (Fig 5)

This will remove all breakpoints from the VBA code, so that you don't have to individually remove each breakpoint, one by one.

Fig 5

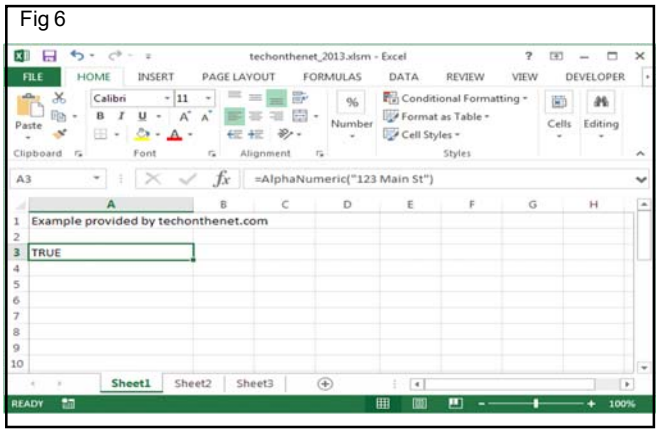


Debug Mode

Now that we know how to set and clear breakpoints in Excel 2010, let's take a closer look at the debug mode in VBA.

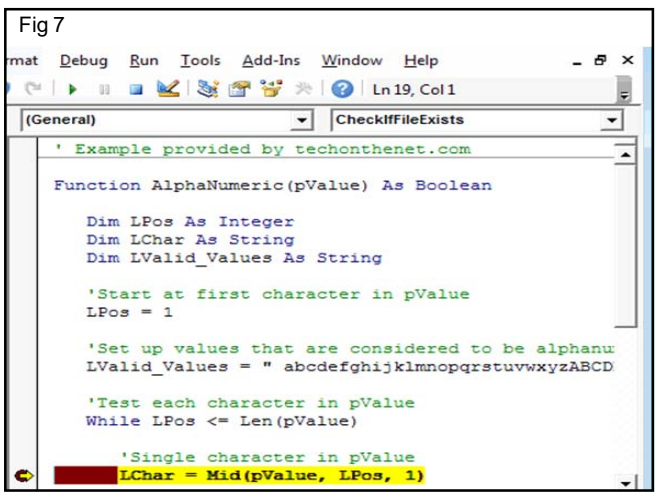
In our example, we've set our breakpoint and entered our AlphaNumeric function as a formula in a cell. This will cause the VBA code to execute. (Fig 6)

Fig 6



When the breakpoint is reached, Excel will display the Microsoft Visual Basic window and highlight the line (in yellow) where the code has been suspended. (Fig 7)

Fig 7



Copyright Free under CC BY Licence

Now we are in debug mode in our Excel spreadsheet. Now we can do any of the following:

1 Check the value of a variable in its current state.

2 Enter VBA code in the Immediate window to view the results.

3 Execute each line of code one at a time.

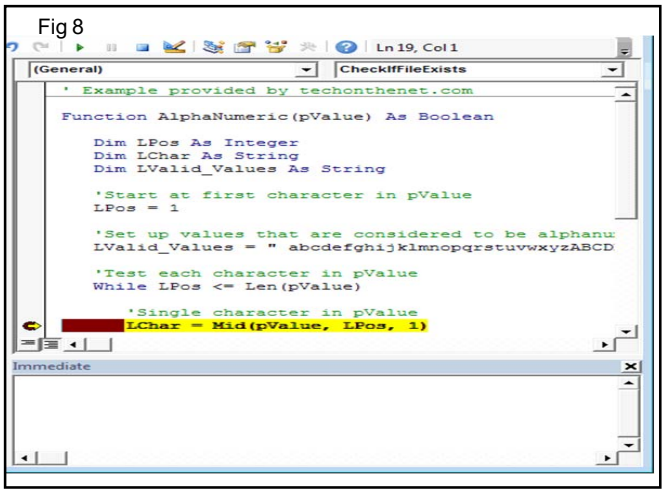
4 Continue execution of the code.

5 Halt execution of the code.

Using the Immediate Window

In Excel 2010, the Immediate window can be used to debug your program by allowing you to enter and run VBA code in the context of the suspended program. (Fig 8)

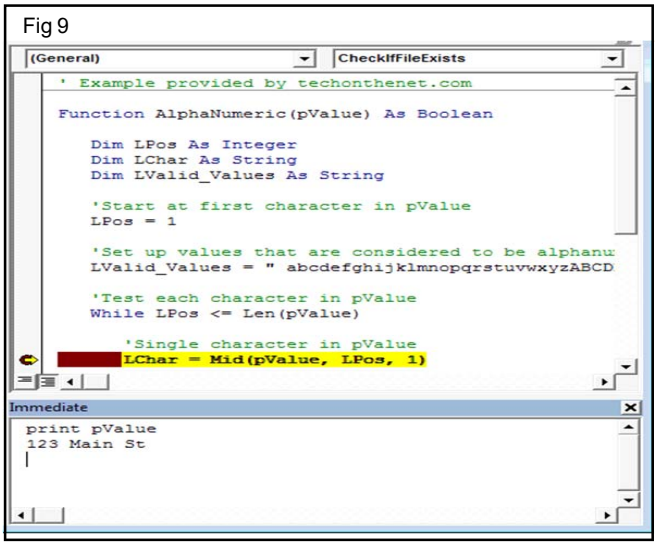
Fig 8



We've found the Immediate window to be the most help when we need to find out the value of a variable, expression, or object at a certain point in the program. This can be done using the print command.

For example, if you wanted to check the current value of the variable called pValue, you could use the print command as follows: (Fig 9)

Fig 9



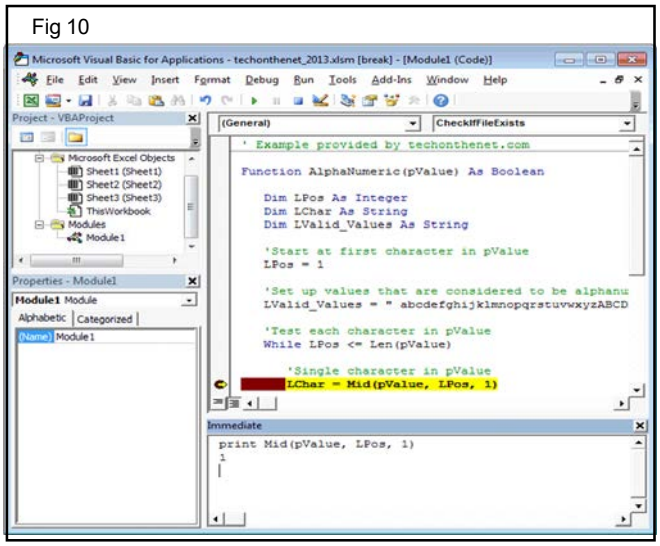
In this example, we typed **print pValue** in the Immediate window and pressed ENTER.

Print pValue

The Immediate window displayed the result in the next line. In this case, the print pValue command returned **123 Main St**.

You can also type more complicated expressions in the Immediate window. (Remember to press ENTER.) For example: (Fig 10)

Fig 10



In this example, we typed **print Mid(pValue, LPos, 1)** in the Immediate window and pressed ENTER.

print Mid(pValue, LPos, 1)

The Immediate window displayed the result of **1** in the next line.

The Immediate window can be used to run other kinds of VBA code, but bear in mind that the Immediate window can only be used when debugging so any code that you run is for debugging purposes only. The code entered in the Immediate window does not get saved and added to the existing VBA code

Adding a Watch Expression

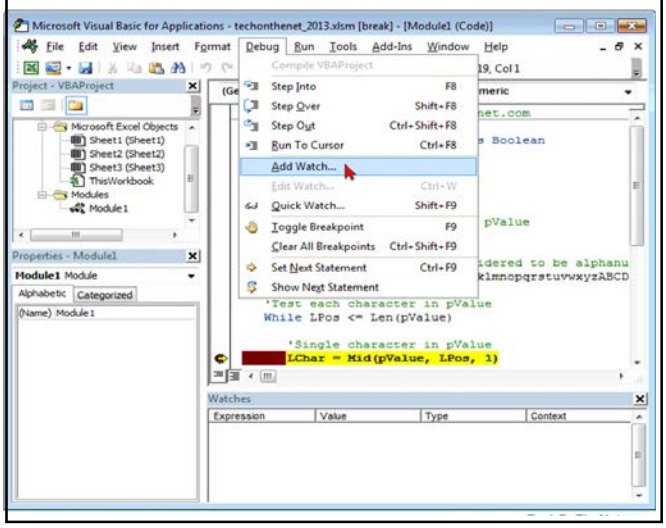
The Watch Window displays the value of a watched expression in its current state. This can be extremely useful when debugging VBA code. Let's explore how to add an expression to the Watch Window.

To add a Watch expression, select **Add Watch** under the **Debug** menu. (Fig 11)

When the *Add Watch* window appears, enter the expression to watch and click the OK button when you are done. (Fig 12)

Copyright Free under CC BY Licence

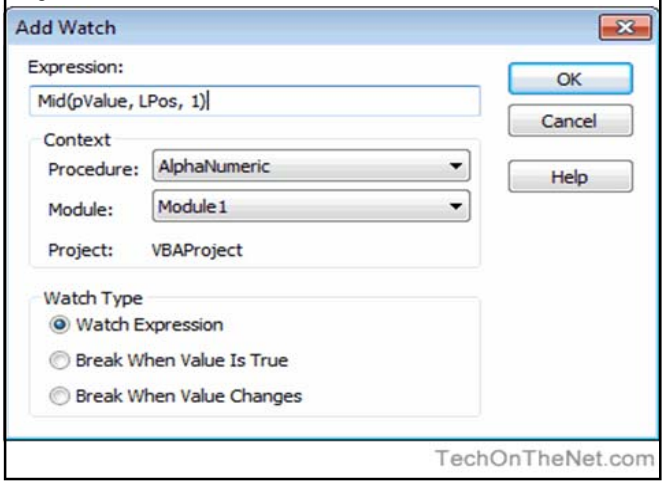
Fig 11



In this example, we've entered the following watch expression in the **Expression** field:

Mid(pValue, LPos, 1)

Fig 12



Next, we've selected AlphaNumeric as the **Procedure** and Module1 as the **Module** when setting up the **Context** for the watched expression.

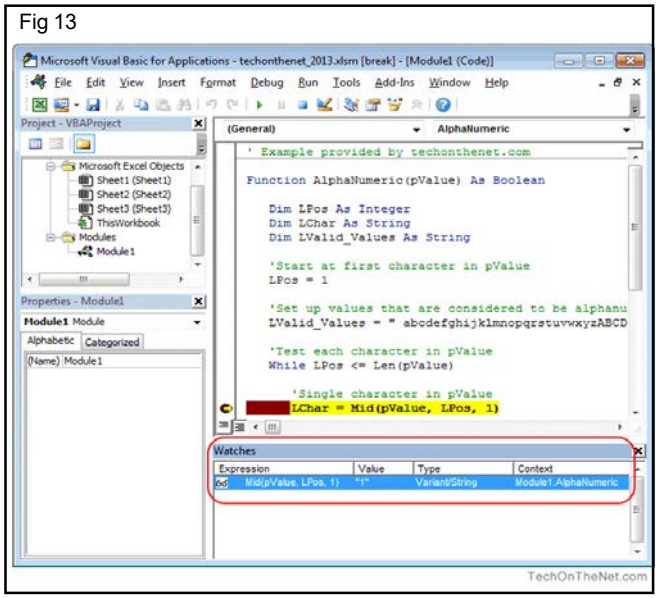
Finally, we've selected *Watch Expression* as the **Watch Type** but there are 3 options to choose from:

Watch Type	Description
Watch Expression	To display the value of the watched expression in its current state
Break When Value Is True	To stop the execution of the code when the value of the watched expression is True
Break When Value Changes	To stop the execution of the code when the value of the watched expression changes

When return to the VBA window, the Watch Window will automatically appear if it was previously hidden. Within the Watch Window, all of the watched expressions should be listed including the one that we just added. (Fig 13)

As you can see, the expression Mid(pValue, LPos, 1) now appears in the Watch Window with a value of "1". Adding a watch is a great way to keep track of variables or expressions of interest when debugging the VBA code.

Fig 13



Copyright Free under CC BY Licence